An ITS Solution Providing Real-Time Visual Overtaking Assistance Using Smartphones

Subhadeep Patra*, Carlos T. Calafate†, Juan-Carlos Cano‡ and Pietro Manzoni§
Department of Computer Engineering,
Universitat Politècnica de València,
Camino de Vera S/N 46022, Valencia, Spain.
Email: *subpat@doctor.upv.es, †calafate@disca.upv.es, ‡jucano@disca.upv.es, §pmanzoni@disca.upv.es

***Abstract*—ITS solutions suffer from the slow pace of adoption by manufacturers despite the interest shown by both consumers and industry. Our goal is to develop ITS applications using already available technologies to make them affordable, quick to deploy, and easy to adopt. In this paper we introduce an ITS system for overtaking assistance that provides drivers with a real-time video feed from the vehicle located just in front. This provides a better view of the road ahead, and of any vehicles travelling in the opposite direction, being especially useful when the front view of the driver is blocked by large vehicles. We evaluated our application using H.264 and MJPEG video encoding formats, and determined the most effective codec choice for our case. Experimental results allow us to be optimistic about the effectiveness and applicability of smartphones in providing overtaking assistance based on video streaming in vehicular networks.**

***Index Terms*—Android application, real implementation, video transmission, live streaming, vehicular network, ITS.**

I. Introduction

Intelligent Transportation Systems (ITS) are advanced solutions that make use of vehicular and infrastructured networks to provide innovative services related to both traffic and mobility management, and that interface with other models of transport. ITS aims at using the already available transport networks in a smarter manner, resulting in significant coordination and safety improvements. Our goal here is to *integrate smartphones into vehicular networks* to develop ITS applications that can reach out to the masses in a short period of time. The choice of smartphones is not only justified by their wide availability and use, but also because they are evolving towards high performance terminals with multi-core microprocessors packed with sufficiently accurate onboard sensors.

The architecture and application has been developed for the Android platform, and requires the devices running it to be equipped with at least a GPS and a back camera. The application makes use of the camera to record video and transmit it over the vehicular network, thus providing an enhanced multimedia information aid for overtaking. The location information of the vehicles gathered from the GPS is useful since the transmission of the video feed only occurs between cars travelling in the same direction, and always occurs from the vehicle in front to the vehicle travelling behind. The Android devices are to be placed on the vehicle dashboard with the camera facing the windshield, so that a clear view of the road in front and cars coming from the opposite direction can be captured. Once started, the application requires no further user interaction to operate, and it can run in the background. Our application can be specially useful in scenarios where the view of the driver is blocked by a larger vehicle, or when a long queue of cars is located ahead and the driver wishes to overtake. In this case, the application will automatically receive the video stream from the vehicle that is leading, and play the received feed on screen, thus aiding the driver in deciding the safest moment to overtake.

We have evaluated the developed application in both indoor and outdoor scenarios. The indoor tests involved comparing the performance of the application using two different video codecs, namely H.264 [15] and MJPEG[1], where the video stream is compressed separately as JPEG [12] images. These two encoding formats were compared focusing mainly on the delay between capture and playback of the video stream. Then, choosing the best codec based on the indoor experiments, we have performed outdoor tests involving real cars. A more detailed explanation about the developed application in terms of the architecture, design, implementation issues and results obtained will be provided in the following sections.

The rest of this paper is organized as follows: In section II, we survey some works in the literature that are closely related to our own. In section III, we will present an overview of the developed application. Later, in section IV, we will present the application modules and some implementation details. The setup used to deploy and validate our application will be described in detail in section V. In section VI, we will present the preliminary results achieved with the application in both a real testbed and a laboratory environment. Finally, section VII concludes this paper summarizing our contributions.

II. Related Works

Both academia and industry have shown a strong interest in the field of ITS, resulting in the development of many innovative applications. Since our application is targeted at smartphones, in this section we are going to focus the bulk of our attention on some of the most interesting smartphone applications related to safe driving.

[1]More on MJPEG: http://en.wikipedia.org/wiki/Motion_JPEG

Most driving safety applications usually aim at warning generation based on onboard location sensors like in the works of Whipple et al. [13], Yang et al. [16], Diewald et al. [3] and Tornell et al. [10]. The application developed by Whipple et al. warns drivers when driving at high speed near schools. Yang et al. concentrated on finding out the probability of accidents based on the location information. DriveAssist, by Diewald et al., triggers warning messages for certain traffic incidents, while Tornell et al., in their proposed application, display on screen important vehicles like ambulances and police cars on a map view, and they later improved that same solution in [8]. Few other applications used the On Board Diagnostics (OBD-II) [4] interface to detect incidents, like in the work of Zaldivar et al. [18] which aimed at detecting accidents. Wideberg et al. [14] also made use of OBD-II devices to extract safety and environment related information.

Only very few applications concentrated on providing visual aids to the drivers, like in SignalGuru described in [6], which leverages collaborative sensing on windshield-mount smartphones, in order to predict the schedule of traffic signals. The CarSafe App [17], introduced by You et al., analyses images from front and back cameras of smartphones to monitor the driver as well as the road ahead. Another interesting application available for download is iOnRoad [1], which aims at providing driving assistance functions including augmented driving, collision warning, and "black-box" like video recording.

Although we have found many different drive safety applications for smartphones, only a handful aimed at providing visual aids to the drivers, namely SignalGuru, CarSafe and iOnRoad. However, none of these smartphone based applications actually provides real-time visual overtaking aids provided by other cars taking advantage of vehicular networks, even though the idea of video-based overtaking assistance systems is not new. Works like the See-Through System [7], which was later improved in [5], although not being targeted for smartphones, are focused on the issue of video-based overtaking assistance. Other related works worth mentioning are [11] and [2], which demonstrate the feasibility of such video-based assistance systems. In [12] authors proposed performance improvements to a video-based overtaking assistant by focusing on codec channel adaptation issues. Whereas, [2] focuses on the reallocation of wireless channel resources to enhance the visual quality.

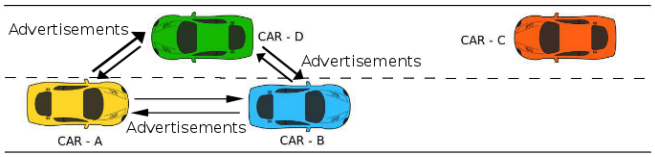
Encouraged by the findings from the above mentioned works, and in order to fulfill the need for a visual overtaking assistance application targeted at the consumers, we decided to develop an application which on being provided with a vehicular network, would require no additional hardware besides a smartphone to operate. The proposed application is targeted at smartphones since we aim at achieving rapid acceptance as well as to promote the close integration of smartphones to vehicular networks.

III. Overview of the Proposed Architecture

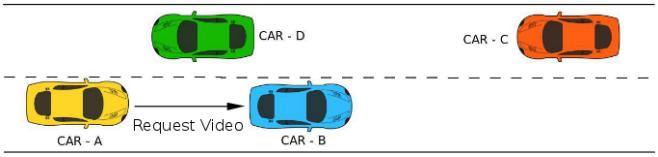
The goal of our application is providing assistance during overtaking by streaming real-time video coming from one vehicle to another. The minimum requirements for running the application is the availability of a device with GPS and a back camera, along with a vehicular network for video transmission.

The functionality of the application can be split in three simple steps for easy understanding. *Step one*, involves *electing the sender and the receiver* of the video which is subject to some special tests and validation conditions. In *step two* the actual *video transmission* occurs between the sender and receiver chosen in phase one. Finally, *step three* is where the application decides to *terminate* the video transmission and playback. This step also involves testing a special condition to stop the video streaming.

In the first step, each device equipped with a back camera and running our application, *broadcasts* an advertisement containing its location and direction, while they are simultaneously listening for incoming broadcast messages coming from other devices. Whenever a device receives broadcast messages from other devices, it first verifies whether the source of the message is valid. This *validity check* is based on tests which basically involve checking if the source and destination vehicles are traveling one ahead of the other, and in the same direction. For a more detailed description of these validation conditions refer to Section IV. If several valid sources are found, the device *requests* video from the best source, which is selected based on the distance between sender and receiver devices. The source vehicle, upon receiving the request to send video from the destination vehicle, starts *streaming* the video signal over the vehicular network, in step two. However, before sending the video, the source double-checks the validation conditions used in step one. The destination vehicle starts playing the video onscreen as soon as it starts receiving it. The streaming and playback process is stopped only when the vehicle behind successfully overtakes, or when it stops following the vehicle in-front, which occurs in step three.



(a) The vehicles exchange *advertisements*.



(b) The client *requests* video from the server.

Fig. 1: Functional overview of the application - *Step one*.

Fig. 1 provides more details about step one. In this example, we have four cars, all of them using our application. CAR-A and CAR-B are travelling in one direction, while CAR-C

and CAR-D travel in the opposite direction. First, the cars broadcast the advertisement to each other as shown in Fig. 1a. Since CAR-C is not within the range of any other car, nobody is able to communicate with it. Each car, upon receiving the advertisement, performs the validity checks to see if the sender of the advertisement is travelling in the same direction and lane. In this case, only CAR-A finds the advertisement message from CAR-B to be valid, and thus requests video from it, as depicted in Fig 1b.

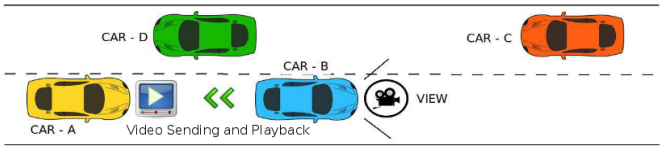


Fig. 2: Functional overview of the application - *Step two*.

Similarly, Fig. 2 shows that CAR-B, upon receiving the video request from CAR-A, rechecks the validity conditions and starts streaming the video. CAR-A starts receiving the video stream and plays it onscreen for its driver. It may be noted here that a device can act both as video source and destination. This is because, while a device is receiving video from another device, it may also be streaming its own video capture to a completely different device.

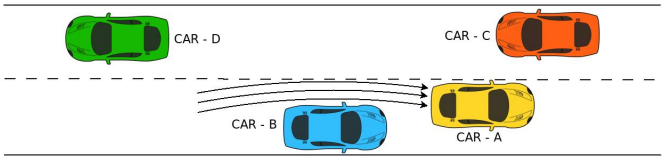


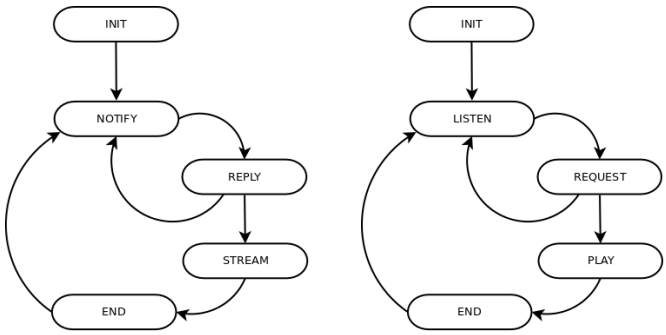
Fig. 3: Functional overview of the application - *Step three*.

Fig. 3, shows that CAR-A has overtaken CAR-B, and this causes the video transmission to stop. Now, CAR-B may request the video feed from CAR-A since it is now travelling ahead, and all the steps above would be repeated in that case.

IV. Implementation Details

From the previous section we already know that the functionality of the developed architecture can be split into three steps. Also, a device running our application can act as both server and client at the same time, receiving video from another device while streaming video to a completely different device. In this section we consider two devices out of which one will be streaming and the other just receiving. The device sending the video is considered as the server, while the receiver act as a client. Despite server and client roles are not yet established at the beginning of *step one*, we will use the words server and client to refer to the devices that will be attaining the respective role in the future for the sake of clarity.

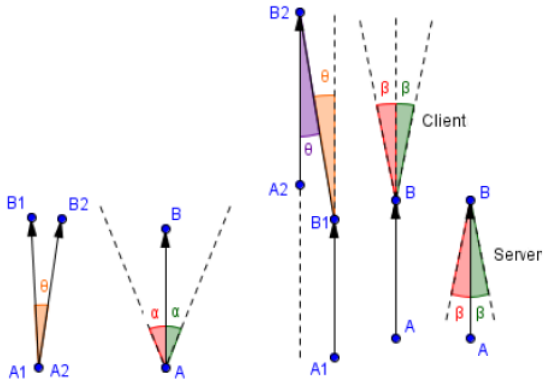
Fig. 4 shows the different states that server and client can attain. When the client and the server start as a part of *step one*, the server is in the *notify* state, as shown in Fig. 4a, and it starts advertising the availability of the video feed by broadcasting a *hello* message. Besides sending advertisements,



(a) Different Server States. (b) Different Client States.

Fig. 4: State diagram of the Server and Client.

the server, while in the *notify* state, also listens for replies to its *hello* message from clients requesting the video feed. A *hello* message contains the location information of the server so that the client, upon receiving it, and by performing some validation tests, can determine if the server is ahead and travelling in the same direction. The client remains listening for advertisements from the server while in the *listen* state, as shown in fig. 4b. If the client receives *hello* messages from different servers, it checks whether the servers are valid, and stores them in a queue of candidate servers. The proposed validity tests include the *same direction test* and the *same lane test* conditions as shown in fig. 5.



(a) Same direction test. (b) Same lane test.

Fig. 5: Validation conditions used to initiate video streaming.

The *same direction test* is used to detect whether two vehicles are travelling in the same direction. For understanding the *same direction test*, let us assume we have two cars, one travelling from the point A1 to B1 and the other from A2 to B2 as shown in Fig. 5a. Notice that, even if two cars are travelling in the same direction and speed, its it hard for them to have an overlapping displacement vector; in other words, the angle between the two vectors is not 0. This can happen due to different driving styles and GPS errors. Thus, we measure the angle θ between these two vectors and compare it to a predefined threshold α. If θ is less than α, we can

Message Type	From → To	Client State	Server State	Message Contents
Hello	S → C	Listen	Notify	Location and Direction
Request	C → S	Request	Notify	Location and Direction
Ready	S → C	Request	Reply	Video sender port
Reject	S → C	Request	Reply	-
Data	S → C	Play	Stream	Location, Direction and Speed
Data-Ack	C → S	Play	Stream	-
End	C → S	Play	Stream	-

TABLE I: Messages exchanged between the Server and Client.

safely assume that the two vehicles are travelling in the same direction. Now, even if two vehicles are travelling in the same direction, it does not necessarily mean that one is ahead of the other, both vehicles may be travelling on different lanes or parallel roads altogether. To check if one is following the other one on the same lane, we perform the *same lane test*, and for this purpose we draw an imaginary line joining the current locations of the two vehicles, as shown in fig. 5b, where B1 and B2 are the current locations. Then we measure the angle of intersection of this line joining the points B1 and B2 with the displacement vectors of the vehicles. When the measured angle of intersection θ is less than a predefined angle β, then the vehicles are considered to be travelling on the same lane. Being on different lanes will result in a higher value of the measured angle θ, and the *same lane test* will fail. If these two conditions are satisfied, then the two vehicles are assumed to be travelling in the same direction, one following the other.

The client which was listening for server advertisements, chooses the best server from the list of candidate servers based on its distance to the server. The client then tries to connect to the chosen server by sending a *request*, and moves to the *request* state. The server, upon receiving the *request* from the client, also checks its validity by performing the *same direction* and *same lane* tests once again. Before sending the *ready* or *reject* message, which denotes whether it is ready to send the video being captured by its camera, the server changes its state to *reply*. The server may further choose to change its state back to *notify* or to *stream* modes depending on its own reply. The client, which was previously in the *request* state, only changes its state to *play* if the reply from the server was a *ready* message containing the *video sender port* number; otherwise it may choose to contact some other server. Table I, details the packet types exchanged between the server and the client.

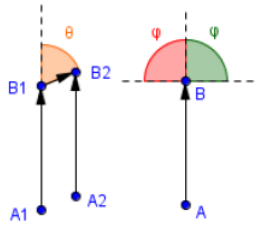


Fig. 6: Overtake test to terminate video streaming.

In case the server and client are in the *stream* and *play* states respectively, *step two*, which involves video streaming and playback, is started at the server and client ends, respectively. Beside streaming video, the server, during this period, keeps sending a *data* message every second. The *data* message contains the location, direction and speed information of the vehicle where the server is located. This way, its corresponding client can check whether an overtake has occurred. To find out if an overtake was successful, the *overtake test* takes place, as shown in fig. 6. This test is similar to the *same lane test*, the only difference being that the angle θ measured here is the other linear pair of the angle of intersection between the displacement vector and the line formed by joining the current location of the two vehicles. Also, the threshold φ used here is usually a much larger value. Upon receiving the *data* message from the server, the client, if it still has not overtaken as suggested by the *overtake test*, replies the server with a *data-ack* message to keep the video connection alive.

If the *overtake test* detects that an overtake has taken place, *step three* takes place, and so the client can request the server to terminate the video stream by sending an *end* message. When the video streaming has been stopped, the client switches to the *end* state and, later on, moves back to the *listen* state once again. The server, on the other hand, can move to the *end* state upon receipt of the *end* message from the client, or if the waiting time for a *data-ack* from the client expires. This waiting time is used to detect cases of eventual disconnections. In our implementation we have fixed this waiting time to 3 seconds, which is adequate to detect disconnections, especially when considering that all communications occur between two cars, one just ahead of the other.

V. Creating the Vehicular Network

For proper operation, the developed application assumes the availability of a vehicular network, although the vehicles we use on a daily basis still lack the capability to communicate with one another. So, for testing our application, we equipped cars with GRCBoxes [9] inside them. GRCBox is a low cost connectivity device based on a *Raspberry Pi*[2] which enables the integration of smartphones into vehicular networks. It was developed mainly due to the difficulty in creating an adhoc network using smartphones. Another important feature provided by GRCBox is the support for V2X communications. The different networks supported by the GRCBox include adhoc, cellular and Wifi access points, among others. Thus,

[2]More on Raspberry Pi: https://www.raspberrypi.org

we use the adhoc network support of the GRCBoxes to create the required network for our application.

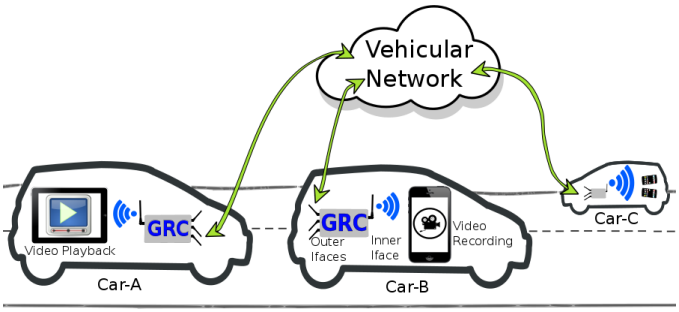


Fig. 7: Our application working together with GRCBox.

Fig. 7 shows how the application works when combined with GRCBox. Each car within the vehicular network has a GRCBox mounted. The smartphones of the passengers within the car are connected to the GRCBox, which is equipped with Wifi-enabled USB interfaces to communicate in adhoc mode, creates a vehicular network. Even though GRCBox is supposed to be equipped with 802.11p for vehicular communication, we used 802.11a devices instead as 802.11p-enabled hardware was not available while setting up the GRCBox to perform the tests. In future experiments we intend to use 802.11p compatible hardware to take advantage of the WAVE standard.

As shown in the figure, Car-B is ahead of the Car-A, and both of them are travelling in the same direction and running our application, so the smartphone in Car-B starts recording the video autonomously and sending it to Car-A, relying on the vehicular network created using the GRCBoxes available within the cars. Concerning the video, it is played onscreen on the device in Car-A as soon as video reception starts.

VI. Application Validation

For validating the application, we performed tests in both indoor and outdoor scenarios. The indoor tests consisted of comparing the delay involved between video capture and its playback, for both H.264 and MJPEG encoding formats. The outdoor tests, on the other hand, involved testing our application and evaluating the various conditions for initiating or terminating video streaming, using real cars driven around the Universitat Politècnica de València. Each car was equipped with a GRCBox to create the required vehicular network, and the Android devices used were a Nexus 7 and a Samsung Galaxy Note 10.1 (2014 Edition). The Nexus 7 from Google was powered by a quad-core 1.2 GHz processor, ULP GeForce GPU, 1 GB RAM and 1.2 MP camera. The Samsung Galaxy Note 10.1, on the other hand, was equipped with a quad-core 1.9 GHz plus quad-core 1.3 GHz processors, 3 GB ram, 8 MP primary camera and 2 MP secondary camera.

A. Delay requirements

The most important factor to determine the proper functioning of a driving assistance application based on streaming real-time video, is the *delay* between video capture and playback. To calculate an admissible value of delay between video capture and playback, let us assume two cars travelling in the opposite direction on a road located in a densely populated area where the possibility of accidents is much higher because roads tend to be more crowded. We know that the maximum speed limit on such roads is usually around 50 km/h. Assuming the worst case where both cars are travelling at maximum speed limit, the *relative velocity* (V_R) can be calculated using the formula:

$$V_R = V_A + V_B$$

where V_A and V_B are the velocities of the two cars, V_R is found to be 100 km/h or 27.778 m/s.

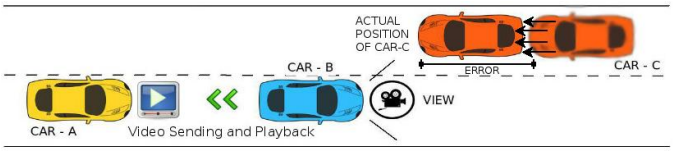


Fig. 8: Error due to delay.

Since there is a delay involved between video capture and playback, the car coming from opposite direction will be in fact closer than the position shown by the application. Fig. 8 demonstrates such a situation, where CAR-A is receiving video feed from CAR-B which shows the position of CAR-C. However, due to the delay involved, CAR-C is located at a position much closer than shown in the video feed. Now, if the allowable error in the position of the vehicle coming from the opposite direction is limited to 10 meters, as displayed by the application, then the *maximum allowable delay* would be *0.36 seconds* in accordance with the equation: $time = distance/speed$. So, in the results that follow, we must make sure that such maximum delay requirement is met.

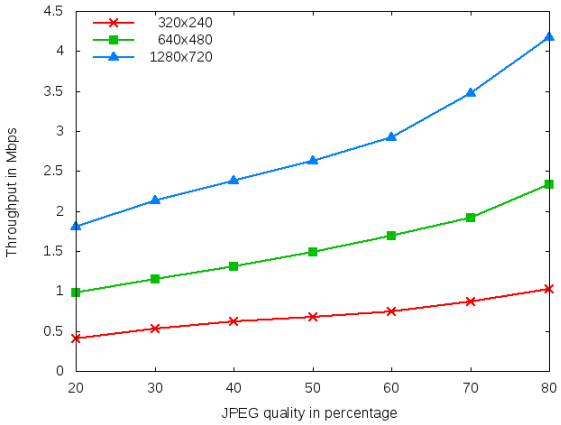
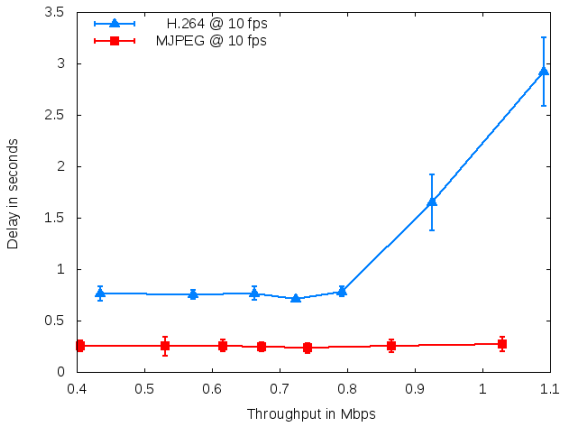


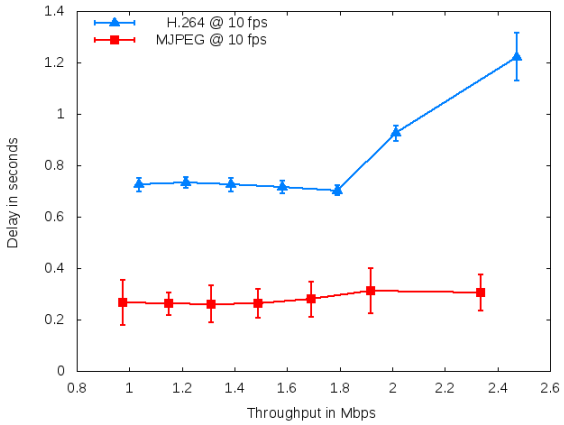
Fig. 9: Variation of throughput with JPEG quality for a 10fps MJPEG video.

B. Indoor tests

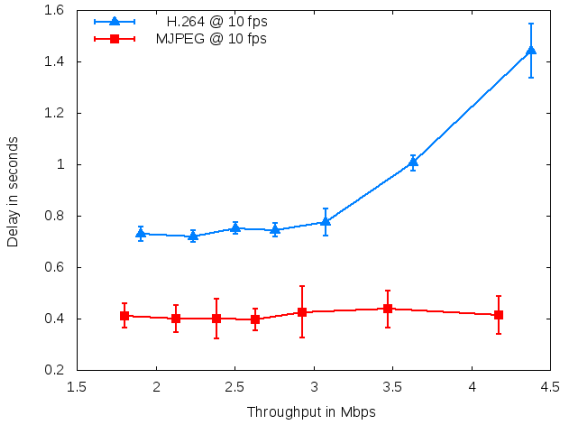
Compared to H.264, MJPEG is a more simpler video compression format since the video stream is compressed sepa-



(a) The delay comparisons for 320x240 video stream.



(b) The delay comparisons for 640x480 video stream.



(c) The delay comparisons for 1280x720 video stream.

Fig. 10: The comparison of MJPEG and H.264.

rately as JPEG images. Thus, when talking about compression-ratios, the performance of MJPEG is limited. So, to make a clear comparison between H.264 and MJPEG encoding schemes for Android devices in terms of delay, we first calculate the throughput of MJPEG video for different resolutions, so that we can eventually make delay versus throughput comparisons for the two encoding formats.

In Fig. 9, the frames per second of the MJPEG video stream was fixed at 10 because we believe that a 10fps video is sufficient for our application. Also, the quality of the JPEG in the video stream was varied from 20 to 80 percent, since for lower values the video quality was too low, whereas a JPEG quality of more than 80 percent did not show any significant improvements in the perceived quality. From the figure we can observe that, for a resolution of 320x240, the average throughput varies from 0.405 to 1.029 Mbps. For 640x480, it lies between 0.976 to 2.336 Mbps, and in case of a 1280x720 resolution, it ranges between 1.805 to 4.177 Mbps.

We now supply the throughput values we achieved for MJPEG to the H.264 encoder to make a proper comparison between them, and to obtain the delay for the two types of encoding formats.

Fig. 10 shows the delay comparison of MJPEG versus H.264 for different resolutions. Fig. 10a allows observing that, for a resolution of 320x240, the average delay for MJPEG suffer minimal variations (from 0.24 to 0.27 seconds), whereas for H.264, it increases from 0.71 to 2.92 seconds. Similarly, fig. 10b shows that the average delay ranges from 0.26 to 0.31 seconds for MJPEG, and from 0.7 to 1.22 seconds for H.264 video, the resolution being 640x480 for both the encoding formats. Eventually, in fig. 10c, which compares H.264 with MJPEG for a resolution of 1280x720, we see that, in case MJPEG is used, the mean delay ranges between 0.4 and 0.44 seconds, being in the range from 0.72 to 1.44 seconds for H.264. Thus, in all the cases, MJPEG outperforms H.264 when considering delay in scenarios involving Android devices. Notice that, although the devices used for our experiments packed sufficient processing power, delay was introduced by the Android libraries. Meaning that MJPEG becomes the wisest choice among the two compression methods.

Next, we want to select the most appropriate resolution and JPEG quality for the MJPEG video stream for use in the scope of our application. The proper functioning of the application is largely dependent on the availability of a vehicular network which has been created using GRCBoxes. Thus, this selection process depends on the bandwidth provided by the vehicular network. From our experiments with the GRCBox, we found that it is capable of providing a mean bandwidth of 10.5Mbps for TCP traffic, and 15.5Mbps for UDP traffic, although the worst value for both TCP and UDP was close to 5.5Mbps. Since, an Android device with our application installed can simultaneously act as video source and destination, the effective bandwidth available for one-way video transmission in the worst case scenario is 2.75Mbps. At the data rate of 2.75Mbps, all the different combinations of resolution up to HD with JPEG quality up to 50 percent, as suggested by the Fig. 9, can be supported by the vehicular network created using GRCBoxes. But, previously we have seen that a delay of more than 360 ms is unacceptable for our real-time visual overtaking aid, consequently we choose to use the MJPEG compression scheme for the resolution of 640x480 at 10fps with JPEG quality set to 80 percent for the video stream, owing to its better performance in terms of delay.

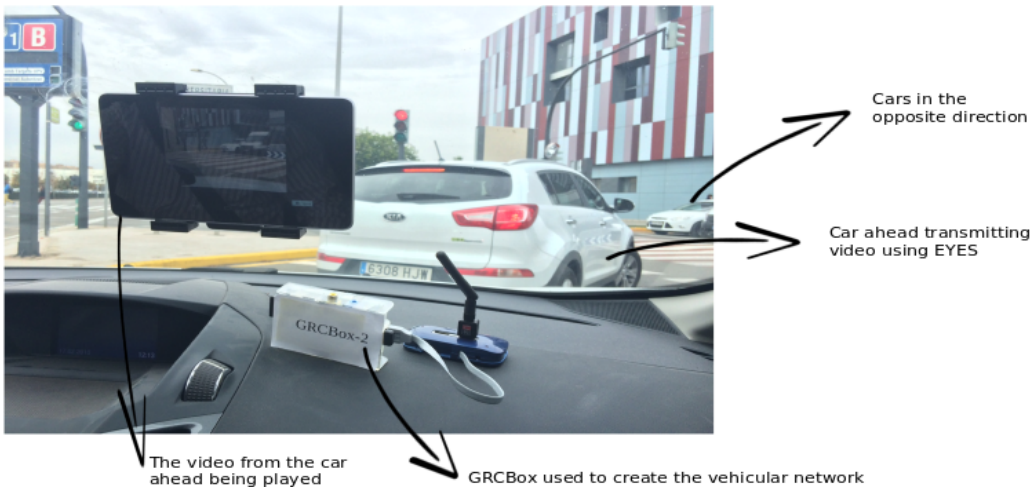


Fig. 11: The experiments with the application in real scenario.

C. Outdoor tests

In our developed architecture, the three important conditions evaluated were described in Section IV, and each of these conditions, namely *same direction test*, *same lane test* and *overtake test*, were dependent on a threshold value. We have performed a wide set of tests in a real scenario, and our aim was to evaluate reasonable values of the threshold angles α, β and φ, for two cars where one follows the other throughout the whole experiment while travelling along a particular route, so that there is non-stop streaming of video between them.

Fig. 11 shows a photo taken during one of the *outdoor tests* [3]. In this picture, we can see that the front car is trying to take a right turn, and the back car is receiving the video from the car ahead and playing it onscreen. While doing our outdoor tests with the application, we collected the various angles used in the three different validation tests. Below we can see the graphical representation of the data obtained during the experiment.

Fig. 12 show the density plot of the angles measured by the *same direction test* at the client side. Most observations for the *same direction test* lies within 20 degrees, which is satisfactory. It is also noticeable that many peaks occur due to GPS errors, also because the route followed had a lot of turns and curves, and so the two cars were not always on a straight path.

Fig. 13 show the density graph of the *same lane test* for the client. From this particular plot, we can see that most observations for the *same lane test* also lie within 20 degrees. Notice that this value is too high considering that this test is very sensitive, and used to detect if cars are travelling on the same lane, and so we find that this condition may not be too useful when considering the accuracy of current technology. Notice that the *same direction test* and *same lane test* are relevant when starting the video streaming, and are evaluated by both the sender and the receiver, but only data from the

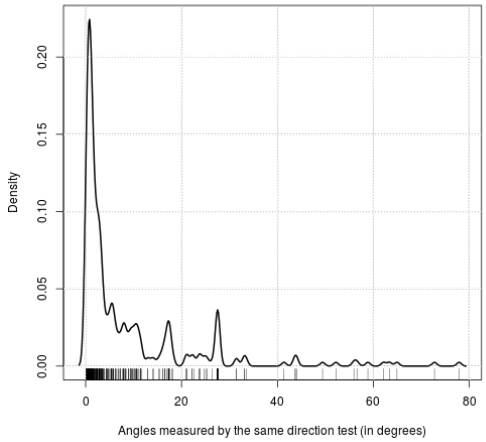


Fig. 12: Results of the *same direction test*.

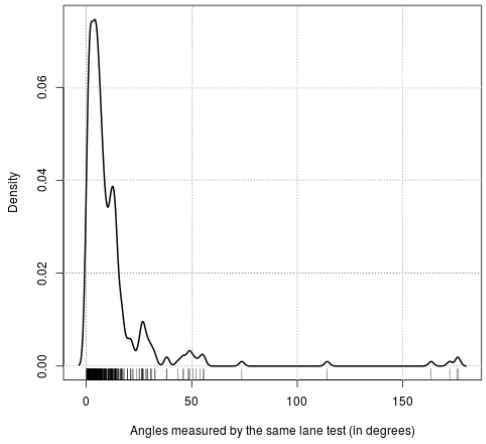


Fig. 13: Results of the *same lane test*.

[3]Application in action: https://www.youtube.com/watch?v=jrIWbFjN3Hw

receiver (i.e. the client) has been plotted in Fig. 12 and 13 as similar values have also been obtained at the server end.

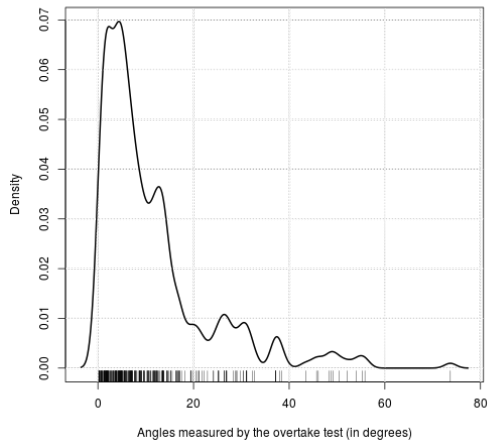


Fig. 14: Results of the *overtake test*.

Fig. 14, shows the density plot for the observations of the *overtake test*. Note that, in order to simplify the graph analysis, the values used are: $180° - \theta$ where θ represents the measured angles in the *overtake test*. The *overtake test* is only evaluated by the client, and we have used its data to produce the graph. We find that the results from this test were pretty much what we expected since all plotted values are below 90 degrees.

VII. Conclusions

In this paper, we have presented a driving safety application that is able to help drivers in safe overtaking. The system provides a real-time video feed captured by the smartphone installed in the vehicle ahead, and which is streamed to the smartphone of the driver seated in the car behind, which displays the video without user intervention. Thus, it provides drivers with important information and helps them to decide whether it is safe to overtake. The developed application was tested using H.264 and MJPEG video encoding formats, and MJPEG was chosen as the default video compression scheme due to its lower encoding delay. We have also evaluated the different test conditions used for starting and stopping autonomous video capture, and found that thresholds of 20 degrees for the *same direction test* and 90 degrees for the *overtake test* are reasonable. Nevertheless, the *same lane test* was found to be useless unless more accurate GPS hardware is made available. Despite this minor issue, we acknowledge the fact that combining smartphones with vehicular networks indeed opens a new horizon for ITS applications and, in the future, we will focus our attention on improving our application by evaluating different alternatives for the *same lane* test, which includes, among others, incorporating image processing techniques for license plate recognition, which can assist the client in choosing the video server.

Acknowledgments

This work was partially supported by the *European Commission* under *Svāgata.eu*, the Erasmus Mundus Programme, Action 2 (EMA2) and the *Ministerio de Economía y Competitividad, Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, Proyectos I+D+I 2014*, Spain, under Grant TEC2014-52690-R.

References

[1] "iOnRoad official website," http://www.ionroad.com/, accessed: 2015-02-8.

[2] E. Belyaev, P. Molchanov, A. Vinel, and Y. Koucheryavy, "The use of automotive radars in video-based overtaking assistance applications," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 14, no. 3, pp. 1035–1042, 2013.

[3] S. Diewald, A. Möller, L. Roalter, and M. Kranz, "DriveAssist-A V2X-Based Driver Assistance System for Android." in *Mensch & Computer Workshopband*, 2012, pp. 373–380.

[4] I. O. for Standardization, "ISO 14230-1:1999: Road vehicles, Diagnostic systems, Keyword protocol 2000," 1999.

[5] P. Gomes, C. Olaverri-Monreal, and M. Ferreira, "Making vehicles transparent through V2V video streaming," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 13, no. 2, pp. 930–938, 2012.

[6] E. Koukoumidis, M. Martonosi, and L.-S. Peh, "Leveraging smartphone cameras for collaborative road advisories," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 5, pp. 707–723, 2012.

[7] C. Olaverri-Monreal, P. Gomes, R. Fernandes, F. Vieira, and M. Ferreira, "The See-Through system: A VANET-enabled assistant for overtaking maneuvers," in *Intelligent Vehicles Symposium (IV), 2010 IEEE*. IEEE, 2010, pp. 123–128.

[8] S. Patra, S. M. Tornell, C. T. Calafate, J.-C. Cano, and P. Manzoni, "Messiah: An ITS drive safety application," in *XXV Jornadas Sarteco, Valladolid, Spain*, 2014.

[9] S. M. Tornell, S. Patra, C. T. Calafate, J.-C. Cano, and P. Manzoni, "GRCBox: Extending Smartphone Connectivity in Vehicular Networks," *International Journal of Distributed Sensor Networks*, 2014.

[10] S. M. Tornell, C. T. Calafate, J.-C. Cano, P. Manzoni, M. Fogue, and F. J. Martinez, "Implementing and testing a driving safety application for smartphones based on the eMDR protocol," in *Wireless Days (WD), 2012 IFIP*. IEEE, 2012, pp. 1–3.

[11] A. Vinel, E. Belyaev, K. Egiazarian, and Y. Koucheryavy, "An overtaking assistance system based on joint beaconing and real-time video transmission," *Vehicular Technology, IEEE Transactions on*, vol. 61, no. 5, pp. 2319–2329, 2012.

[12] G. K. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30–44, 1991.

[13] J. Whipple, W. Arensman, and M. S. Boler, "A public safety application of GPS-enabled smartphones and the android operating system," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. IEEE, 2009, pp. 2059–2061.

[14] J. Wideberg, P. Luque, and D. Mantaras, "A smartphone application to extract safety and environmental related information from the OBD-II interface of a car," *International Journal of Vehicle Systems Modelling and Testing*, vol. 7, no. 1, pp. 1–11, 2012.

[15] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H. 264/AVC video coding standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, 2003.

[16] J. Yang, J. Wang, and B. Liu, "An intersection collision warning system using Wi-Fi smartphones in VANET," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*. IEEE, 2011, pp. 1–5.

[17] C.-W. You, N. D. Lane, F. Chen, R. Wang, Z. Chen, T. J. Bao, M. Montes-de Oca, Y. Cheng, M. Lin, L. Torresani *et al.*, "Carsafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM, 2013, pp. 13–26.

[18] J. Zaldivar, C. T. Calafate, J.-C. Cano, and P. Manzoni, "Providing accident detection in vehicular networks through OBD-II devices and Android-based smartphones," in *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*. IEEE, 2011, pp. 813–819.